# Clase 3 — 06.11.25

## 🧠 Programación de Servicios y Procesos

👨‍🏫 Profesor: José Antonio Martín
Unidad: Programación Multiproceso
🗓️ Clase 3 — 06/11/2025
🎯 Tema: Procesos (Planificación y Algoritmos)

## 1️⃣4️⃣ NIVELES DE PLANIFICACIÓN

La planificación se divide en tres niveles porque la gestión de procesos actúa sobre distintos momentos del ciclo de vida del proceso. Cada planificador maneja eventos distintos y afecta a diferentes estados del sistema.


### 🔹 Planificación a corto plazo (Short-Term Scheduler)

Es el planificador más activo y determinante. Opera con alta frecuencia, normalmente cada pocos milisegundos, porque es el encargado de asignar la CPU.


**Responsabilidades clave:**

- Mantiene la **CPU ocupada** el máximo tiempo posible.
- Selecciona el proceso ideal de la cola **Ready/Preparado**.
- Ejecuta las transiciones: `Listo → En ejecución En ejecución → Listo En ejecución → Bloqueado`
- Determina la expulsión de un proceso cuando:
  - expira el quantum,
  - llega un proceso con mayor prioridad,
  - hay interrupciones externas (I/O, timer, signals).

**Impacto en el sistema:**

- Controla la **sensación de velocidad** del sistema.
- Afecta a la experiencia del usuario (responsividad).
- Influye en el número de cambios de contexto por segundo.
- Afecta directamente la eficiencia del dispatcher.

Si este planificador es eficiente, la CPU se mantiene ocupada ejecutando trabajo útil en lugar de estar idle.

## 🔹 Planificación a medio plazo (Mid-Term Scheduler)

Actúa a una escala mayor y su impacto se nota en la **utilización de memoria** y en la **multiprogramación**.

Su función principal es **regular cuántos procesos están activos simultáneamente en memoria principal (RAM).**

**Problemática habitual:**

- Si muchos procesos están bloqueados por E/S, la CPU queda infrautilizada.
- Si todos los procesos están en RAM aunque no ejecuten, la memoria queda saturada.

Este planificador detecta esos escenarios y aplica **swapping**.

**Operaciones:**

- Mueve procesos bloqueados a memoria secundaria (disco) → *suspensión*.
- Recupera procesos suspendidos cuando vuelven a estar en condiciones de ejecutarse → *reanudar*.

**Beneficios:**

- Libera RAM para procesos activos.
- Evita que el sistema entre en un estado donde “todo está bloqueado”.
- Mantiene el nivel adecuado de multiprogramación.
- Reduce la presión sobre el planificador a corto plazo.

Es un mecanismo de control de carga que estabiliza el sistema.

## 🔹 Planificación a largo plazo (Long-Term Scheduler)

Este planificador controla el **ritmo de admisión de nuevos procesos al sistema**.

Su objetivo no es la CPU directamente, sino el **flujo de trabajo global** que entra desde almacenamiento secundario hacia memoria.

**Funciones principales:**

- Decide cuántos procesos nuevos pueden entrar al sistema.
- Admite trabajos desde disco y los convierte en procesos activos.
- Mantiene un equilibrio entre procesos CPU-bound e I/O-bound.

**Importancia del equilibrio:**

- Demasiados CPU-bound → la CPU se satura y los procesos interactivos sufren.
- Demasiados I/O-bound → la CPU queda infrautilizada mientras todos esperan por E/S.

El planificador a largo plazo asegura que exista una combinación saludable de ambos tipos.

**Actúa en tiempos más largos:**

- segundos
- minutos
- incluso más en sistemas batch o servidores

---

## ✅ Cómo trabajan juntos los tres niveles

Los niveles no son independientes; forman una cadena de control:

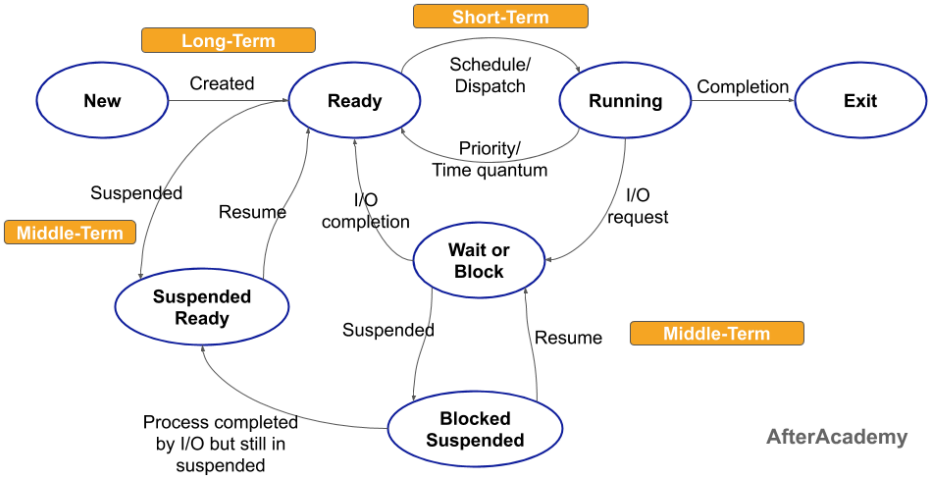
1. **Largo plazo** decide cuántos trabajos entran al sistema ➜ regula el volumen.
2. **Medio plazo** decide cuáles están en memoria en cada instante ➜ regula la carga.
3. **Corto plazo** decide quién usa la CPU ➜ regula el rendimiento inmediato.

**Interacciones típicas:**

- Largo plazo alimenta la cola de listos.
- Medio plazo asegura que los procesos alimentados al corto plazo están realmente en RAM y listos.
- Corto plazo selecciona entre ellos de forma eficiente.

---

## ✅ Resultado global




Un sistema operativo moderno necesita estos tres niveles para:

- evitar saturación de memoria
- evitar CPU idle cuando hay trabajo pendiente
- evitar “tormentas” de procesos nuevos
- mantener rendimiento y respuesta
- garantizar equidad
- minimizar ineficiencias y bloqueos

En conjunto, forman el cerebro de la gestión de procesos en sistemas multiprogramados.

---

# 1️⃣5️⃣ PLANIFICACIÓN A CORTO PLAZO

El planificador a corto plazo se encarga de escoger el próximo proceso que ejecutará la CPU. Sus decisiones impactan directamente en el rendimiento visible para el usuario.

## Qué hace exactamente

- Examina los procesos en la cola de listos.
- Selecciona el proceso según el algoritmo activo.
- Da la orden al dispatcher para que cargue su contexto.
- Interrumpe procesos que exceden su **quantum**.
- Reactiva procesos prioritarios si llegan.

## Relación con el rendimiento

El profesor muestra el Administrador de Tareas en Windows para evidenciar:

- % de CPU en uso.
- Este valor refleja cuán eficiente es el planificador a corto plazo.

Si hay muchos ciclos libres, la CPU está infrautilizada.

---

# 1️⃣6️⃣ PLANIFICACIÓN A LARGO PLAZO

Actúa sobre la entrada de nuevos trabajos al sistema —generalmente desde almacenamiento secundario — y se usa en sistemas que manejan **lotes de trabajos** además de procesos interactivos.

## Objetivos

- Mantener equilibrada la cantidad de procesos.
- Introducir trabajos cuando la CPU está poco ocupada.
- Evitar saturar la memoria con demasiados procesos.
- Mantener ocupados los recursos en momentos de baja actividad.

## Casos de uso

Procesos por lotes:

- tareas rutinarias
- cálculos masivos
- informes
- automatizaciones

Ideal para periodos de baja interacción del usuario.

---

# 1️⃣7️⃣ PLANIFICACIÓN A MEDIO PLAZO

En sistemas de multiprogramación, varios procesos compiten por la memoria principal. Cuando los procesos están bloqueados, la CPU puede quedar inutilizada.

## Problema

Puede suceder que:

- Todos los procesos en memoria estén bloqueados por E/S.
- Ninguno esté listo.
- La CPU no tenga trabajo.

## Solución

El planificador a medio plazo realiza swapping:

- **Memoria principal → secundaria** para suspender procesos.
- **Secundaria → principal** para reanudarlos cuando los recursos están listos.

Así evita que el sistema quede inactivo y mejora el uso de CPU.

---

# 1️⃣8️⃣ TIPOS DE ALGORITMOS DE PLANIFICACIÓN

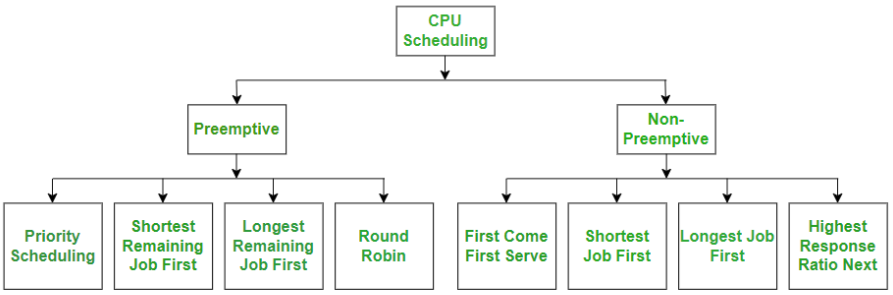
Los algoritmos de planificación definen la **estrategia concreta** que usa el planificador para decidir qué proceso obtiene la CPU en cada instante. La elección del algoritmo afecta directamente a:

- latencia percibida por el usuario,
- número de cambios de contexto,
- eficiencia de CPU,
- riesgo de inanición,
- equidad entre procesos.

Por eso se dividen en dos categorías principales.

---

## ✅ Cooperativos (non-preemptive)




El sistema **no interrumpe** un proceso una vez ha empezado a ejecutarse, salvo que el propio proceso:

- termine,
- o se bloquee en una operación de E/S.

Características clave:

- Control simple del planificador.
- Menor sobrecarga porque hay menos cambios de contexto.
- Comportamiento predecible.

Problema estructural:

- **Efecto convoy:**
  un proceso largo “arrastra” a todos los demás.
  Ejemplo: P1 = 50 ms, P2 = 2 ms, P3 = 1 ms.
  P2 y P3 deben esperar a que P1 termine, aunque sean cortos.

Ventajas:

- Simplicidad de implementación.
- Ideal para sistemas muy básicos o con carga baja.

Desventajas:

- No responde bien a sistemas interactivos.
- Penaliza procesos cortos y sensibles.

---

## ✅ Apropiativos o expulsivos (preemptive)

El planificador puede **expulsar** un proceso incluso si no ha terminado.

Motivos típicos:

- expiración del quantum,
- llegada de un proceso más prioritario,
- interrupciones externas,
- eventos de hardware/temporizador.

Ventajas:

- **Mejor tiempo de respuesta**, especialmente en sistemas interactivos.
- Evita inanición porque el planificador fuerza rotaciones.
- Permite implementar prioridades dinámicas.

Costes:

- Más cambios de contexto → mayor overhead.
- Mayor complejidad de implementación.
- Debe mantener estimaciones de tiempo y prioridades.

Uso práctico:

- Sistemas multitarea modernos.
- Sistemas operativos de escritorio.
- Servidores Linux con CFS.

---

# 1️⃣9️⃣ MÉTRICAS DE PLANIFICACIÓN

Los algoritmos se evalúan mediante métricas cuantitativas. Cada métrica refleja un aspecto diferente del rendimiento del sistema.

## 🔹 Tiempo de espera (Waiting Time, E)

Tiempo acumulado que un proceso pasa en la cola de **listos** sin usar CPU.

Impacto:

- Determina la equidad del sistema.
- Afecta notablemente a procesos cortos.
- Se usa para detectar inanición.

---

## 🔹 Tiempo de retorno (Turnaround Time)

Tiempo total desde que el proceso **entra en el sistema** hasta que **finaliza**.

Incluye:

```
espera + ejecución + E/S + cambios de contexto
```

Es una métrica global. Mide el tiempo efectivo de completar un trabajo.

---

## 🔹 Tiempo de respuesta (Response Time)

Tiempo desde que un proceso entra en la cola de listos hasta que se ejecuta **por primera vez**.

Importancia:

- Es crítico en interfaces gráficas y sistemas interactivos.
- Afecta directamente a la percepción del usuario.

Ejemplo claro:

- Abrir una aplicación.
- Cambiar de pestaña.
- Responder a un input.

---

## 🔹 Uso de CPU (CPU Utilization)

Porcentaje de tiempo en que la CPU está ejecutando procesos útiles.

Objetivo:

- Maximizarlo sin saturar el sistema.

Una CPU con uso del 60–90% suele indicar un sistema sano.

---

## 🔹 Rendimiento (Throughput)

Número de procesos completados por unidad de tiempo.

Favorecido por:

- algoritmos eficientes,
- baja sobrecarga,
- menos cambios de contexto.

---

## 🔹 Equidad (Fairness)

Garantiza que **ningún proceso** quede permanentemente sin CPU.

Factores que la afectan:

- prioridades mal configuradas,
- falta de rotación,
- algoritmos no expulsivos,
- sistemas con demasiada multiprogramación.

---

**Relación entre métricas:**

- Minimizar tiempo de espera no siempre implica maximizar rendimiento.
- Un quantum demasiado corto mejora la respuesta pero empeora el uso de CPU.
- Demasiados procesos pueden saturar cola de listos y reducir equidad.

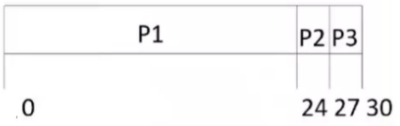
Una planificación eficiente busca un equilibrio entre todas las métricas según las necesidades del sistema.

---

# 2️⃣0️⃣ FCFS (First Come First Served / FIFO)

El algoritmo FCFS es el más simple de todos: los procesos se atienden estrictamente en el orden en que llegan. Es el equivalente a hacer una cola en una ventanilla: el primero en llegar es el primero en ser atendido, sin interrupciones y sin prioridades.

| Procesos | llegada | Tiempo uso CPU |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 2 | 3 |
| P3 | 4 | 3 |




Tiempo de espera: (0+22+23)/3=18 dependiente del orden de llegada (Efecto convoy)
Equitativo.

## 🔍 Características detalladas

- **No expulsivo**
  Una vez un proceso entra a CPU, permanece allí hasta que finaliza o se bloquea.
  No se interrumpe aunque llegue uno más corto o más urgente.
- **Fácil de implementar**
  Solo necesita una cola FIFO. Sin cálculos de prioridad ni estimaciones de duración.
- **Equitativo en orden de llegada**
  No favorece ni penaliza a nadie en función de tamaño o prioridad.
- **Problema central: efecto convoy**
  Si el primer proceso es largo (CPU-bound), obliga a todos los demás a esperar.
  Es el equivalente a un camión lento bloqueando el tráfico.

FCFS (Example)

| Process | Duration | Oder | Arrival Time |
|---|---|---|---|
| P1 | 24 | 1 | 0 |
| P2 | 3 | 2 | 0 |
| P3 | 4 | 3 | 0 |

**Gantt Chart :**


P1 waiting time : 0
P2 waiting time : 24
P3 waiting time : 27

The Average waiting time :
(0+24+27)/3 = 17

---

## 🧩 Ejemplo

Procesos:

| Proceso | Llegada | Duración CPU |
|---|---|---|
| P1 | 0 | 24 |
| P2 | 2 | 3 |
| P3 | 4 | 3 |

Orden de ejecución (diagrama de Gantt):

```
P1 | P2 | P3 0 24 27 30
```

---

## 🧱 Tiempo de espera explicado

Tiempo de espera = tiempo que un proceso pasa en la cola de listos.

- P1: espera 0
  Entra y ejecuta directamente.
- P2: espera = fin de P1 - llegada P2 = 24 - 2 = 22
- P3: espera = fin de P2 - llegada P3 = 27 - 4 = 23

Media = (0 + 22 + 23) / 3 = **18**

## 🧠 Conclusión

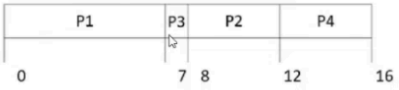
FCFS penaliza procesos cortos porque siempre tendrán que esperar a los largos que llegaron antes. Es eficiente solo si todos los procesos tienen duraciones similares.

---

# 2️⃣1️⃣ SJF (Shortest Job First)

SJF selecciona siempre el proceso con la **menor duración total estimada**. Su objetivo es reducir el tiempo medio de espera.

| Procesos | llegada | Tiempo uso CPU |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

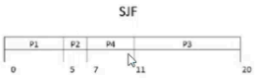



- Menor tiempo de espera: (0+3+6+7)/4=4
- No equitativo: los procesos más largos se pueden quedar permanentemente sin CPU (**inanición**)
- Complejo: requiere predicción y ajuste

39

| Procesos | llegada | Tiempo uso CPU |
|---|---|---|
| P1 | 0 | 5 |
| P2 | 1 | 2 |
| P3 | 2 | 9 |
| P4 | 3 | 4 |


Tiempo de espera: (0+4+5+13)/4=5, 5




Tiempo de espera: (0+4+4+9)/4=4,25

## 🔍 Características detalladas

- **No expulsivo**
  Una vez selecciona un proceso, lo ejecuta hasta el final.
- **Minimiza el tiempo promedio de espera**
  Matemáticamente es óptimo bajo ciertas condiciones.
- **Estimación del tiempo de ejecución**
  Esta es la parte complicada: el SO debe predecir cuánto tardará un proceso, cosa que no siempre es trivial.
- **Riesgo de inanición**
  Si llegan muchos procesos cortos, los largos podrían quedar esperando indefinidamente.

---

## 🧩 Ejemplo

Procesos:

- P1 = 7

- P2 = 4
- P3 = 1
- P4 = 4

Orden:

`P3 → P2 → P4 → P1`

Los más cortos primero, luego los largos.

## 🧠 Conclusión

SJF es muy eficiente cuando los tiempos son conocidos. Pero en entornos generales, el sistema no puede prever exactamente la duración de un proceso.

---

# 2️⃣2️⃣ SRTF (Shortest Remaining Time First)

SRTF es la **versión preemptive de SJF**. Aquí la duración restante de cada proceso es la clave.

| Procesos | llegada | Tiempo uso CPU |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |



- Tiempo de espera: (9+1+0+2)/4=3
- Tiempo de respuesta: (0+0+0+2)/4=0,5
- Requiere predicción y ajuste.

## 🔍 Características detalladas

- **Expulsivo**
  Si llega un proceso con menor tiempo restante, se interrumpe el proceso actual automáticamente.
- **Excelente tiempo de respuesta**
  Perfecto para trabajos cortos y urgentes.
- **Alta complejidad**
  Necesita actualizar constantemente la estimación de tiempo restante.
- **Mayor sobrecarga**
  Genera más cambios de contexto porque interrumpe procesos frecuentemente.

## 🧩 Ejemplo

Imagina estos procesos:

| Proceso | Llegada | CPU total |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

## ⏱️ Evolución del sistema segundo a segundo

### 0–2: P1

Solo existe P1, así que empieza. Le quedan 7 → 5.

### 2–4: P2

P2 llega con tiempo total 4.
Comparación: P1=5 restantes, P2=4 → SRTF elige **P2**.
Le quedan 4 → 2.

### 4–5: P3

Llega P3 con tiempo total 1.
Comparación: P1=5, P2=2, P3=1 → **entra P3**.
Ejecuta 1 unidad y termina.

### 5–7: P2

Tras P3, el que menos resta es P2 (2).
Ejecución → P2 termina.

### 7–11: P4

P4 llega con 4 unidades.
Comparación: P1=5, P4=4 → ejecuta P4 hasta terminar 5–9.
(Según los tiempos de ejemplo del profesor, los valores pueden variar)

### Fin: P1

Queda P1 con 5 unidades.
Se ejecuta hasta terminar.

## 🔍 Gantt simplificado

```
P1 | P2 | P3 | P2 | P4 | P1
0   2   4   5   9  16
```

## 🧮 Tiempo de espera (promedio)

- P1: espera mientras se ejecutan P2, P3, P4 → 9
- P2: espera desde llegada 2 → 1
- P3: llega 4 y se ejecuta inmediato → 0
- P4: llega 5, espera hasta 9 → 4

Media = (9 + 1 + 0 + 4) / 4 = **3,5**

## 🧠 Conclusión

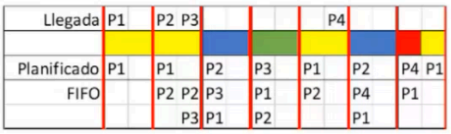
Fantástico para minimizar tiempos de espera, pero costoso en términos de gestión y cambios de contexto.

---

# 2️⃣3️⃣ Round Robin (RR)

Round Robin es el algoritmo expulsivo más usado en sistemas interactivos. Está diseñado para mantener equidad y respuesta rápida.

| Procesos | llegada | Tiempo uso CPU |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 3 | 2 |
| P4 | 9 | 1 |




- Tiempo de espera: (7+6+3+3)/4=4,75
- Tiempo de respuesta: (0+2+3+3)/4=2

## 🔍 Funcionamiento explicado

- Cada proceso recibe un **quantum** (tiempo fijo).
- Se ejecuta durante ese quantum.
- Si no termina, se expulsa y vuelve al final de la cola.
- La cola sigue un orden FIFO.

Es como repartir turnos iguales en una mesa redonda.

## 🧩 Ejemplo

Quantum = 2

Procesos:

| Proceso | Llegada | CPU |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 3 | 2 |
| P4 | 9 | 1 |

## ⏱️ Ejecución paso a paso

### 0–2: P1

Quedan → 5

### 2–4: P2

Quedan → 2

### 4–6: P1

Quedan → 3

### 6–8: P2

P2 termina

### 8–10: P1

Quedan → 1

### 10–12: P3

P3 termina

### 12–13: P1

P1 termina

### 13–14: P4

P4 termina

### 🔍 Gantt simplificado

```
P1 | P2 | P1 | P2 | P1 | P3 | P1 | P4 0 2 4 6 8 10 12 13 14
```

### 🧮 Tiempo de espera

Se calcula sumando todo el tiempo que cada proceso pasa en cola de listos:

- P1: 7
- P2: 6
- P3: 3
- P4: 3

Media = **4,75**

---

## ✅ Ventajas

- **Alta equidad**: todos los procesos reciben tiempo.
- **Buena respuesta**: ideal para sistemas interactivos (SO de escritorio).
- **Evita inanición**: siempre acaba tocándole el turno a cada proceso.

---

## ❌ Desventajas

- **Quantum muy bajo**
  Demasiados context switches → sobrecarga → menos trabajo útil.
- **Quantum muy alto**
  Pierde la ventaja del reparto justo → se comporta como un FCFS.

---

## 🧠 Conclusión

RR es un compromiso entre equidad y eficiencia. Funciona especialmente bien en entornos con muchos procesos interactivos y tiempos impredecibles.

- SRTF = mejor espera promedio, peor overhead.
- RR = mejor equidad y respuesta, espera media aceptable pero no óptima.

---

## 2️⃣4️⃣ EJERCICIO PRÁCTICO: ROUND ROBIN (Quantum = 2)

Simulación realizada por el profesor en bloc de notas.

Datos:

- P1: llegada 0, uso 7
- P2: llegada 2, uso 4
- P3: llegada 3, uso 2
- P4: llegada 9, uso 1

Quantum = 2

### Secuencia completa

```
0–2  → P1
2–4  → P2
4–6  → P1
6–8  → P2 (termina)
8–10 → P1
10–12 → P3 (termina)
12–13 → P1 (termina)
13–14 → P4 (termina)
```

Orden final:
**P1 → P2 → P1 → P2 → P1 → P3 → P1 → P4**

---

# 2️⃣5️⃣ TIEMPO DE ESPERA EN ROUND ROBIN

En Round Robin, el tiempo de espera de cada proceso se calcula sumando **todos los periodos** en los que el proceso está en la cola de listos sin ejecutarse. Dado que RR expulsa procesos de la CPU repetidamente, cada proceso alterna entre pequeñas ráfagas de ejecución y múltiples intervalos de espera.

Valores calculados:

- **P1: 7**
- **P2: 6**
- **P3: 3**
- **P4: 3**

Esto incluye la suma de:

- esperas iniciales,
- esperas entre ráfagas,
- esperas generadas por la llegada de procesos posteriores,
- reordenamiento natural de la cola FIFO.

## Media = 4,75

---

## 🧠 Análisis del resultado

El resultado del tiempo de espera medio muestra una tendencia característica de RR:

- **RR reparte el tiempo de CPU de forma mucho más equilibrada que FCFS o SJF.**
- Ningún proceso se queda “atascado” detrás de un proceso largo durante todo su tiempo de ejecución.
- La espera está **fragmentada** en pequeños tramos, lo que mejora la percepción de “progreso” en sistemas interactivos.

Sin embargo:

- RR **no minimiza** el tiempo de espera; solo lo distribuye mejor.
- La eficiencia depende del tamaño del quantum.
- Demasiadas interrupciones añaden overhead, lo que puede aumentar el tiempo total real de finalización de los procesos.

Conclusión clave:

> Round Robin mejora la equidad y la percepción de respuesta, pero sacrifica eficiencia óptima en tiempo de espera.

---

# 2️⃣6️⃣ IMPACTO DEL TAMAÑO DEL QUANTUM

El quantum (Q) es el parámetro crítico del algoritmo Round Robin. Controla cuánto tiempo puede ejecutar un proceso antes de ser expulsado.

---

## 🔹 Quantum corto (Q muy pequeño)

- El planificador interrumpe procesos **muy frecuentemente**.
- Gran número de **cambios de contexto**.
- La CPU pierde tiempo guardando y restaurando estados.
- Aumenta el overhead del sistema.
- Tiempo de respuesta excelente (ideal para entornos interactivos).
- Pero el rendimiento total baja porque la CPU gasta demasiado tiempo en gestión interna.

**Resultado:**
Sistema muy reactivo, pero menos eficiente.

---

## 🔹 Quantum largo (Q grande)

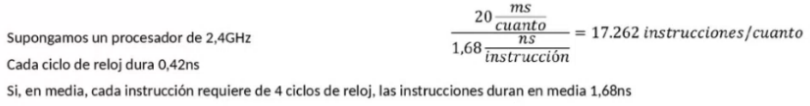
- Pocos cambios de contexto.
- Mejor aprovechamiento del tiempo de CPU.
- Menor overhead.
- Peor tiempo de respuesta.
- La equidad se reduce.
- Los procesos cortos pueden esperar demasiado.

Si Q es tan grande como la duración media de los procesos ➜ **Round Robin se convierte en FCFS**.

**Resultado:**
Sistema eficiente desde la perspectiva de CPU, pero con mala experiencia interactiva.

---

## 🧩 Ejemplo de la diapositiva




CPU: **2.4 GHz**
Quantum: **20 ms**

Cálculo:

- 2.4 GHz = 2.400.000.000 ciclos por segundo
- 20 ms = 0,02 s
- Ciclos en 20 ms = 2.4e9 × 0.02 = 48.000.000 ciclos

Si cada instrucción tarda 1,68 ns → unas **17.262 instrucciones por quantum**.

Interpretación real:

- Un quantum de 20 ms es suficiente para ejecutar miles de instrucciones.
- Para un usuario, 20 ms es imperceptible → respuesta fluida.
- Para la CPU, 20 ms es mucho tiempo → pocos cambios de contexto.

**Equilibrio excelente para sistemas de escritorio.**

---

# 2️⃣7️⃣ PROCESOS EN LINUX

En Linux, un proceso es un programa en ejecución junto con su contexto completo:

- PID,
- memoria asignada,
- ficheros abiertos,
- entorno,
- señalización.

Linux gestiona procesos de forma muy eficiente, permitiendo miles de procesos simultáneos gracias a:

- planificadores avanzados (CFS),
- control granular de prioridades,
- separación de hilos y procesos,
- namespaces y cgroups para contenedores.

---

## 🔧 Comandos útiles

### `top`

- Muestra procesos en tiempo real.
- Indica:
  - %CPU,
  - consumo de RAM,
  - estados,
  - prioridades,
  - load average.

### `ps`

- Lista procesos en diferentes formatos.
- Combinado con flags (`auxf`, `ajx`) ofrece vistas completas.

---

## 🧠 ¿Por qué Linux domina en servidores?

- Maneja muchos procesos con alta eficiencia.
- Permite **control granular** del sistema:
  - prioridades,
  - afinidad de CPU,
  - limitación de recursos.

- Mantiene servicios activos durante años sin reiniciar.
- Permite contenedores (Docker, Kubernetes).
- Escala bien en hardware multiprocesador y multinúcleo.

Aproximadamente un **90%** de servidores de Internet usan Linux por su estabilidad y capacidad de gestión de procesos.

---